Design Doc: Implementing Unpacked Extensions Listing via CDP

Attention: Externally visible, non-confidential
Author: nharshunova@chromium.org
Status: Draft
Short Link: go/chrome-devtools:list-extensions-cdp-command-design
Created: 2026-02-02 / **Last Updated**: 2026-02-03

One-page overview

Summary

This document details the implementation of a new Chrome DevTools Protocol (CDP) command, `Extensions.getExtensions`. This command is designed to retrieve a complete list of installed unpacked extensions, including metadata like the local file path. Its introduction provides the foundational layer for automated end-to-end testing of extensions.

Platforms

Desktop

Team

nharshunova@chromium.org

Tracking issue

https://crbug.com/477183926

Value proposition

Implementing the `getExtensions` CDP command is essential for providing automated e2e tools with core extension management capabilities. By exposing the complete unpacked extension list metadata (ID, name, version, enabled status, and path), it unblocks the implementation of various management features, such as reloading extensions, by allowing tools to programmatically identify specific extensions.

Code affected

CDP, Extensions

Signed off by

Name	Approval status
Alex Rudenko (chrome-devtools-mcp TL)	LGTM
Danil Somsikov (CDP)	LGTM
Alex Gough (Chrome Security)	LGTM

User stories

User goal:

As a Chrome extension developer, I want to be able to debug the extensions I am developing. Therefore, I need an API that will list all the unpacked extensions in my browser. This simplifies the debugging process by allowing automations to be able to identify specific extensions and perform actions on them (like triggering or reloading).

Specific clients

Puppeteer, chrome-devtools-mcp, WebDriver

Design

The `getExtensions` command is a new method added to the Extensions domain of the Chrome DevTools Protocol. It is designed to provide the necessary extension metadata in a single, efficient call to external automation tools like `chrome-devtools-mcp`.
CDP Command Structure (PDL):

The command definition is structured to return an array of `ExtensionInfo` objects:

```
None
  # Detailed information about an extension.
  type ExtensionInfo extends object
    properties
      # Extension id.
      string id
      # Extension name.
      string name
      # Extension version.
      string version
```

```
      # Extension path.
      string path
      # Extension enabled status.
      boolean enabled

  command getExtensions
    returns
      array of ExtensionInfo extensions
```

The implementation for the command involves retrieving the `ExtensionRegistry` from the browser context. It uses the internal `ExtensionRegistry::GenerateInstalledExtensionsSet()` function to gather all installed extensions. The logic then iterates through this set and filters for unpacked extensions by checking the extension's location with `extensions::Manifest::IsUnpackedLocation(extension->location())`. Only extensions that meet the unpacked criteria are packaged into the ExtensionInfo result object. The inclusion of the `path` property provides the possibility to correlate between runtime extension IDs and local directories for managing **unpacked** extensions (e.g., reloading after local code changes).

Alternatives considered

One alternative considered was programmatically parsing the `chrome://extensions` page to extract the necessary metadata. However, this was deemed an unreliable and non-viable solution as it would require the management tools to rely heavily on the specific HTML markup and internal implementation details of the page, which are subject to changes. There is currently no other single CDP command that aggregates all the essential extension metadata.

Rollout plan

Waterfall

Core principle considerations

Speed

Unlike other ExtensionsHandler APIs, the new `getExtensions` command only iterates over the in-memory `ExtensionRegistry` to collect the data and does not perform any disk I/O actions. No significant negative impact on Chrome's speed, memory usage, or power consumption is expected.

Security

The `getExtensions` command exposes sensitive metadata, most notably the **absolute local file system paths** of installed extensions. To mitigate the risk of unauthorized information disclosure, this command enforces the same strict security model as other privileged `Extensions` domain commands like `loadUnpacked`, `uninstall`, and `triggerAction`.

Consistency in the `ExtensionsHandler` is maintained by utilizing the `allow_loading_extensions` check, which restricts access to trusted clients meeting the following criteria:

- The client is considered trusted (i.e., `channel->GetClient()->AllowUnsafeOperations()`).
- The browser was launched with the `--enable-unsafe-extension-debugging` command-line flag.
- The request originates specifically from a browser-level agent (i.e., `agent_host->GetType() == content::DevToolsAgentHost::kTypeBrowser`).

Simplicity

The design introduces a single, isolated CDP command, ensuring minimal impact on the overall complexity of the DevTools protocol. The implementation reuses existing internal extension data, providing a clean and straightforward interface for external automation tools.

Privacy

The command exposes sensitive information about the installed extensions. This is mitigated by the following:

- The same information is already exposed to CDP clients via the chrome://extensions page.
- The info is only available to the local caller who has access to the user data dir already.
- The information only included unpacked extensions installed in the developer mode.

Testing plan

1. Implement tests to verify that `Extensions.getExtensions` correctly retrieves all installed unpacked extensions with accurate metadata (ID, name, version, enabled status, and local file path).
2. Verify that the command properly distinguishes between unpacked and packed extensions, returning only the former.

Follow-up work

1. Implement support for the new CDP command in `Puppeteer`.
2. Implement the corresponding *list_extensions* tool within the `chrome-devtools-mcp` repository, which will utilize the new Puppeteer function.